Ex.No: 1	**BASIC DATA TYPES**
Date :	

Aim

To write a Python program that demonstrates the usage of different basic data types such as int, float, complex, string, list, tuple, dictionary, and Boolean.

Algorithm

Step 1: Start the program.

Step 2: Declare variables of different basic data types:

- Integer
- Float
- Complex
- String
- List
- Tuple
- Dictionary
- Boolean

Step 3: Display the values of each variable.

Step 4: Display the data type of each variable using the type() function.

Step 5: End the program.

Program

```
# Program to demonstrate different basic data types in Python

# Integer
a = 10

# Float
b = 10.5
```

```
# Complex
c = 2 + 3j

# String
d = "Python Programming"

# List
e = [10, 20, 30, 40]

# Tuple
f = (1, "Hello", 5.6)

# Dictionary
g = {"name": "Priya", "age": 21, "course": "Python"}

# Boolean
h = True

# Display values and their types
print("Value of a =", a, "| Type =", type(a))
print("Value of b =", b, "| Type =", type(b))
print("Value of c =", c, "| Type =", type(c))
print("Value of d =", d, "| Type =", type(d))
print("Value of e =", e, "| Type =", type(e))
print("Value of f =", f, "| Type =", type(f))
print("Value of g =", g, "| Type =", type(g))
print("Value of h =", h, "| Type =", type(h))
```

Output

Inference:

Viva-Voce:

1) **What is the difference between a list and a tuple in Python?**

2) **How does Python distinguish between an integer and a float data type?**

3) **What is a complex data type in Python and how is it represented?**

4) **What is a dictionary in Python and how does it differ from a list?**

5) **What is the Boolean data type in Python and where is it commonly used?**

Result:

Ex.No:2	**ARITHMETIC OPERATIONS.**
Date :	

Aim

To develop a menu-driven program in Python that reads input from the console and performs different arithmetic operations (addition, subtraction, multiplication, division, and modulus) based on the user's choice.

Algorithm

Step 1: Start the program.

Step 2: Display the menu with the list of arithmetic operations:

- Addition
- Subtraction
- Multiplication
- Division
- Modulus
- Exit

Step 3: Read the user's choice.

Step 4: If the choice is between 1 and 5:

- Read two numbers from the user.
- Perform the selected arithmetic operation.
- Display the result.

Step 5: If the choice is 6, exit the program.

Step 6: If the user enters an invalid choice, display an error message.

Step 7: Repeat the menu until the user selects Exit.

Step 8: End the program.

Program

```
# Menu driven arithmetic operations program

while True:
    print("\n----- Arithmetic Operations Menu -----")
    print("1. Addition")
    print("2. Subtraction")
    print("3. Multiplication")
    print("4. Division")
    print("5. Modulus")
    print("6. Exit")

    choice = int(input("Enter your choice (1-6): "))

    if choice == 6:
        print("Exiting the program... Thank You!")
        break

    # Taking input only if choice is in valid range
    if choice in [1, 2, 3, 4, 5]:
        num1 = float(input("Enter first number: "))
        num2 = float(input("Enter second number: "))

    if choice == 1:
        print("Result:", num1 + num2)

    elif choice == 2:
        print("Result:", num1 - num2)

    elif choice == 3:
        print("Result:", num1 * num2)

    elif choice == 4:
        if num2 != 0:
            print("Result:", num1 / num2)
        else:
            print("Error: Division by zero is not allowed!")

    elif choice == 5:
        if num2 != 0:
            print("Result:", num1 % num2)
        else:
            print("Error: Modulus by zero is not allowed!")
```

```
    else:
        print("Invalid choice! Please enter a number between 1 and 6.")
```

Output

Case 1	Case 2	Case 3

Inference:

Viva-Voce:

1) **What is a menu-driven program and why is it used?**

2) **How does Python take input from the user through the console?**

3) **Which control structure is commonly used to implement a menu in Python and why?**

4) **What happens if division by zero is attempted in a Python program?**

5) What is the difference between the division (/) operator and the modulus (%) operator in Python?

Result:

Ex.No:3	DECISION AND LOOPING STATEMENTS
Date :	

Aim

To write a Python program using decision statements to check whether a given number is positive, negative, or zero.

Algorithm

Step 1: Start the program.
Step 2: Read a number from the user.
Step 3: If the number is greater than 0, display "Positive".
Step 4: Else if the number is less than 0, display "Negative".
Step 5: Else display "Zero".
Step 6: Stop the program

Program

```
num = float(input("Enter a number: "))

if num > 0:
    print(num, "is Positive")
elif num < 0:
    print(num, "is Negative")
else:
    print("The number is Zero")
```

Output

Enter a number: -6
-6.0 is Negative

Aim

To write a Python program using a looping statement (while loop) to calculate the factorial of a given number.

Algorithm

Step 1: Start the program.
Step 2: Read a number from the user.
Step 3: Initialize factorial = 1 and counter i = 1.
Step 4: Repeat while i ≤ number:

- Multiply factorial by i

- Increment i by 1

Step 5: Display the factorial result.
Step 6: Stop the program.

Program

```
num = int(input("Enter a number: "))
fact = 1
i = 1

while i <= num:
    fact = fact * i
    i += 1

print("Factorial of", num, "=", fact)
```

Output

Output 1	Output 2	Output 3

Inference:

Viva-Voce:

1) What are decision statements in Python?

2) Which decision statement is most suitable for checking multiple conditions and why?

3) How does Python evaluate a condition in an if statement?

4) What data type is returned by a comparison operation in Python?

5) Why is it important to check for zero separately when classifying a number?

Result:

Ex.No:4	**BUILT-IN STRING HANDLING FUNCTIONS**
Date:	

Aim

To write a Python program that demonstrates the usage of various built-in string handling functions such as len(), upper(), lower(), title(), strip(), replace(), split(), startswith(), endswith(), and find().

Algorithm

Step 1: Start the program.

Step 2: Read a string input from the user.

Step 3: Apply various built-in string functions on the given string:

- Find the length of the string.
- Convert the string to uppercase.
- Convert the string to lowercase.
- Convert the string to title case.
- Remove leading and trailing spaces.
- Replace a substring with another substring.
- Split the string into a list of words.
- Check if the string starts or ends with a specific substring.
- Find the position of a substring.

Step 4: Display the results of each function.

Step 5: End the program.

Program

```
# Program to demonstrate various built-in string handling functions
# Reading input from the user
text = input("Enter a string: ")

print("\n--- String Handling Functions Output ---")
print("Original String:", text)
print("Length of string:", len(text))
print("Uppercase:", text.upper())
print("Lowercase:", text.lower())
print("Title Case:", text.title())
print("Strip (removes spaces):", text.strip())
print("Replace 'a' with '@':", text.replace("a", "@"))
print("Split words:", text.split())
print("Starts with 'Hello'? :", text.startswith("Hello"))
print("Ends with 'Python'?  :", text.endswith("Python"))
print("Find position of 'is':", text.find("is"))
```

Output

Inference:

Viva-Voce:

1) **What are built-in string functions in Python?**

2) **What is the purpose of the strip() function?**

3) **What is the difference between find() and index() string functions?**

4) **How does the split() function work in Python strings?**

5) **What is the use of upper() and lower() string functions?**

Result:

Ex.No:5	**STRING OPERATIONS WITHOUT BUILT-IN FUNCTIONS**
Date :	

Aim

To write a Python program to implement various string operations such as finding length, string copy, string concatenation, string comparison, and string reversal without using built-in string handling functions.

Algorithm

Step 1: Start the program.

Step 2: Read two strings as input from the user.

Step 3: To find the length of a string:

Initialize a counter to 0 and increment it for each character in the string using a loop.

Step 4: To copy a string:

Create an empty string and append each character from the original string using a loop.

Step 5: To concatenate two strings:

Create an empty string and append characters of the first string followed by characters of the second string.

Step 6: To compare two strings:

Compare characters one by one; if all characters match, the strings are equal, otherwise not.

Step 7: To reverse a string:

Loop through the string from the last character to the first and append it to a new string.

Step 8: Display the results of all operations.

Step 9: Stop the program.

Program

```
# Program to demonstrate string operations without using built-in functions

s1 = input("Enter first string: ")
s2 = input("Enter second string: ")

# 1. Find length
def string_length(s):
    count = 0
    for _ in s:
        count += 1
    return count

# 2. Copy string
def string_copy(s):
    copy = ""
    for ch in s:
        copy += ch
    return copy

# 3. Concatenate two strings
def string_concat(a, b):
    result = ""
    for ch in a:
        result += ch
    for ch in b:
        result += ch
```

```
    return result

# 4. Compare two strings
def string_compare(a, b):
    if string_length(a) != string_length(b):
        return False
    for i in range(string_length(a)):
        if a[i] != b[i]:
            return False
    return True

# 5. Reverse string
def string_reverse(s):
    rev = ""
    length = string_length(s)
    for i in range(length - 1, -1, -1):
        rev += s[i]
    return rev
```

Output

Inference:

Viva-Voce:

1) **What does it mean to perform string operations without using built-in functions?**

2) How can the length of a string be calculated without using len()?

3) How can two strings be concatenated manually in Python?

4) What logic is used to reverse a string without built-in functions?

5) Why is learning string manipulation without built-in functions important?

Result:

Ex.No:6a)	**USER-DEFINED FUNCTIONS WITH FUNCTION ARGUMENTS**
Date:	

Aim

To write a Python program that implements a simple calculator using user-defined functions to perform addition, subtraction, multiplication, and division.

Algorithm

Step 1: Start the program.

Step 2: Define four functions: add(), sub(), mul(), and div() which take two parameters and return the result.

Step 3: Display a menu to the user to choose an arithmetic operation.

Step 4: Read two numbers from the user.

Step 5: Based on user choice, call the required function and display the result.
Step 6: If the user selects an invalid choice, display a message.
Step 7: End the program.

Program

```
# Simple Calculator using user-defined functions

def add(a, b):
    return a + b

def sub(a, b):
    return a - b

def mul(a, b):
    return a * b

def div(a, b):
    if b == 0:
        return "Error! Division by Zero"
    return a / b

print("----- Simple Calculator -----")
print("1. Addition\n2. Subtraction\n3. Multiplication\n4. Division")

choice = int(input("Enter your choice (1-4): "))
n1 = float(input("Enter first number: "))
n2 = float(input("Enter second number: "))

if choice == 1:
    print("Result =", add(n1, n2))
elif choice == 2:
    print("Result =", sub(n1, n2))
elif choice == 3:
    print("Result =", mul(n1, n2))
elif choice == 4:
    print("Result =", div(n1, n2))
else:
    print("Invalid Choice")
```

Output

Inference:

Viva-Voce:

1) **What is a user-defined function in Python?**

2) **Why are functions used in a calculator program?**

3) **How are arguments passed to a function in Python?**

4) **What happens if division by zero occurs in the division function?**

5) What is the advantage of using separate functions for each arithmetic operation?

Result:

Ex.No:6b)	**PASS BY VALUE AND PASS BY REFERENCE.**
Date :	

Aim

To write a Python program using user-defined functions that demonstrates pass by value and pass by reference in Python.

Algorithm

Step 1: Start the program.

Step 2: Create a function that takes an integer as a parameter and modifies it inside the function (immutable type – acts like pass by value).

Step 3: Create a function that takes a list and modifies it (mutable type – acts like pass by reference).

Step 4: Call both functions and display the values before and after the function call.

Step 5: Observe how changing parameters inside the function affects values outside the function.

Step 6: End the program.

Program

```
# Program to demonstrate pass by value & pass by reference

# Function with immutable type (pass by value behavior)
def modify_value(x):
    x = x + 10
    print("Inside modify_value(), x =", x)

# Function with mutable type (pass by reference behavior)
def modify_list(lst):
    lst.append(100)
    print("Inside modify_list(), lst =", lst)

# Immutable object
num = 50
print("Before calling modify_value(), num =", num)
modify_value(num)
print("After calling modify_value(), num =", num)

print("\n------------------------------------")

# Mutable object
numbers = [10, 20, 30]
print("Before calling modify_list(), numbers =", numbers)
modify_list(numbers)
print("After calling modify_list(), numbers =", numbers)
```

Output

Inference:

Viva-Voce:

1) **What is meant by pass by value and pass by reference?**

2) **How does Python handle parameter passing for immutable and mutable data types?**

3) Why do changes to a list inside a function affect the original list outside the function?

4) How can pass by value behavior be simulated in Python?

5) In which cases is pass by reference preferred over pass by value?

Result:

Ex.No:7a)	**LINEAR SEARCH AND BINARY SEARCH USING LIST**
Date:	

Aim

To write Python programs to search an element from a list using Linear Search and Binary Search.

Algorithm (Linear Search)

Step 1: Start the program.

Step 2: Read the list elements and the key element to be searched.

Step 3: Traverse the list from the first element to the last.

Step 4: Compare each element with the key.

Step 5: If a match is found, return the index position.

Step 6: If the loop ends without finding the key, report that the element is not found.

Step 7: End the program.

Algorithm (Binary Search)

Step 1: Start the program.

Step 2: Read the sorted list elements and the key to be searched.

Step 3: Initialize low = 0 and high = length – 1.

Step 4: Repeat until low ≤ high:Compute mid = (low + high) // 2

If key == list[mid], search is successful.

If key < list[mid], update high = mid – 1.

Else update low = mid + 1.

Step 5: If low > high, report that the element is not found.

Step 6: End the program.

Program

```
# Linear Search function
def linear_search(lst, key):
    for i in range(len(lst)):
        if lst[i] == key:
            return i
    return -1

# Binary Search function
def binary_search(lst, key):
    low = 0
    high = len(lst) - 1
    while low <= high:
        mid = (low + high) // 2
        if lst[mid] == key:
            return mid
        elif key < lst[mid]:
            high = mid - 1
        else:
            low = mid + 1
    return -1

# Main program
lst = list(map(int, input("Enter elements of the list separated by space: ").split()))
key = int(input("Enter the element to search: "))

print("\n--- Linear Search ---")
pos = linear_search(lst, key)
print("Element found at index", pos if pos != -1 else "Element not found")

print("\n--- Binary Search (List will be sorted) ---")
lst.sort()
pos = binary_search(lst, key)
print("Element found at index", pos if pos != -1 else "Element not found")
```

Output

Inference:

Viva-Voce:

1) **What is linear search and how does it work on a list?**

2) **What is binary search and what is its basic requirement?**

3) **Why must the list be sorted before applying binary search?**

4) **What is the time complexity of linear search and binary search?**

5) In which situations is linear search preferred over binary search?

Result:

Ex.No:7b)	**MATRIX OPERATIONS USING NESTED LIST**
Date:	

Aim

To write a Python program to perform matrix operations (addition, subtraction, and multiplication) using nested lists.

Algorithm

Step 1: Start the program.

Step 2: Read number of rows and columns.

Step 3: Read elements of Matrix A and Matrix B using nested lists.

Step 4: For addition:

Add corresponding elements of A and B and store result in a new matrix.

Step 5: For subtraction:

Subtract corresponding elements of A and B and store result in a new matrix.

Step 6: For multiplication:

For each element in the resultant matrix, multiply and sum row × column values.

Step 7: Display all operation results.

Step 8: End the program.

Program

```
# Matrix Operations using Nested List
```

```python
r = int(input("Enter number of rows: "))
c = int(input("Enter number of columns: "))

print("Enter elements of Matrix A:")
A = [[int(input()) for j in range(c)] for i in range(r)]

print("Enter elements of Matrix B:")
B = [[int(input()) for j in range(c)] for i in range(r)]

# Matrix Addition
add = [[A[i][j] + B[i][j] for j in range(c)] for i in range(r)]

# Matrix Subtraction
sub = [[A[i][j] - B[i][j] for j in range(c)] for i in range(r)]

# Matrix Multiplication
mul = [[0 for j in range(c)] for i in range(r)]
for i in range(r):
    for j in range(c):
        for k in range(c):
            mul[i][j] += A[i][k] * B[k][j]

print("\nMatrix A =", A)
print("Matrix B =", B)
print("\nAddition =", add)
print("Subtraction =", sub)
print("Multiplication =", mul)
```

Output

Inference:

Viva-Voce:

1) **What is a matrix in Python and how is it represented using nested lists?**

2) **How do you access an element of a matrix stored as a nested list?**

3) **What conditions must be satisfied to perform matrix addition and subtraction?**

4) **What is the rule for matrix multiplication using nested lists?**

5) **Why are nested loops required for implementing matrix operations in Python?**

Result:

Ex.No:8	TUPLE
Date :	

Aim

To write a Python program to create a tuple and perform the following operations:

- Add items to a tuple
- Find the length using len()
- Check whether an item exists in a tuple
- Access items in a tuple

Algorithm

Step 1: Start the program.

Step 2: Create a tuple with predefined elements.

Step 3: To add items, convert the tuple to a list, append the item, and convert back to a tuple (since tuples are immutable).

Step 4: Use len() function to find the number of elements in the tuple.

Step 5: Check whether an element exists in the tuple using the in keyword.

Step 6: Access tuple items using indexing.

Step 7: Display all results.

Step 8: Stop the program.

Program

```
# Create a tuple
my_tuple = (10, 20, 30, 40, 50)
print("Original Tuple:", my_tuple)

# 1) Add item to tuple (Tuples are immutable, so convert to list)
item = int(input("Enter an item to add: "))
temp_list = list(my_tuple)
temp_list.append(item)
my_tuple = tuple(temp_list)
print("Tuple after adding item:", my_tuple)

# 2) len() - find length of tuple
print("Length of tuple:", len(my_tuple))

# 3) Check whether item exists in tuple
check_item = int(input("Enter an item to search in tuple: "))
if check_item in my_tuple:
    print(check_item, "is present in tuple")
else:
    print(check_item, "is NOT present in tuple")

# 4) Access items
print("First item:", my_tuple[0])
print("Last item:", my_tuple[-1])
print("Items from index 1 to 3:", my_tuple[1:4])
```

Output

Inference:

Viva-Voce:

1) **Is it possible to add items directly to a tuple? Explain why or why not.**

2) **How is the length of a tuple determined in Python?**

3) **How can you check whether a specific item exists in a tuple?**

4) **How are elements accessed in a tuple?**

5) **Why are tuples preferred over lists in certain situations?**

Result:

Ex.No:9	**DICTIONARY**
Date:	

Aim

To create a dictionary in Python and perform various dictionary operations such as printing items, accessing items, using get (), changing values, and using len ().

Algorithm

Step 1: Start the program.

Step 2: Create a dictionary with key–value pairs (e.g., roll number and name).

Step 3: Display the dictionary items using print() and items().

Step 4: Access items individually using keys.

Step 5: Use the get() method to access a value for a specified key.

Step 6: Change the value of an existing key in the dictionary.

Step 7: Use the len() function to find the total number of items in the dictionary.

Step 8: Display the updated dictionary.

Step 9: Stop the program.

Program

```
# Dictionary Operations

# Step 1: Create a dictionary
student = {"Roll_No": 101, "Name": "Priya", "Course": "Python"}

# Step 2: Print dictionary items
print("Dictionary Items:", student)

# Step 3: Access items
print("Access Roll_No:", student["Roll_No"])
print("Access Name:", student["Name"])

# Step 4: Use get() method
print("Access Course using get():", student.get("Course"))

# Step 5: Change values
student["Name"] = "Anitha"
print("After modifying Name:", student)

# Step 6: Use len()
print("Length of Dictionary:", len(student))
```

Output

Inference:

Viva-Voce:

1) **What is a dictionary in Python and how is it different from a list or tuple?**

2) **How can dictionary items be accessed using keys?**

3) **What is the purpose of the get() method in a dictionary?**

4) **How can the value of an existing key be changed in a dictionary?**

5) **What does the len() function return when applied to a dictionary?**

Result:

Ex.No:10	**A PACKAGE, A SUB-PACKAGE AND MODULES**
Date :	

Aim

To write a Python program to create a package named college, a sub-package named alldept, and modules for AI_DS, CSE, AIML, and CSBS, and to implement admin() and cabin() functions in each module.

Algorithm

1) Start the program.
2) Create a main package directory named college.
3) Inside the college package, create a sub-package directory named alldept.
4) Add an __init__.py file in both college and alldept directories to make them Python packages.
5) Inside the alldept sub-package, create module files for each department:
    a. AI_DS.py
    b. CSE.py
    c. AIML.py
    d. CSBS.py
6) In each module, define two functions:
    a. admin() to display administrative details of the department
    b. cabin() to display cabin or staff room details of the department

7) Create a main Python program outside the package to import the required modules from college.alldept.
8) Call the admin() and cabin() functions from each module to demonstrate package, sub-package, and module usage.
9) Display the result.
10) End the program.

Program Files

```
# Package: college
# Sub-package: all_dept
# Modules: AI_DS, CSE, AIML, CSBS

# -------- AI & DS Module --------
def ai_ds_admin():
    print("AI & DS Department Admin: Dr. Meena")

def ai_ds_cabin():
    print("AI & DS HOD Cabin: Block A – Room 203")


# -------- CSE Module --------
def cse_admin():
    print("CSE Department Admin: Dr. Prakash")

def cse_cabin():
    print("CSE HOD Cabin: Block B – Room 110")


# -------- AIML Module --------
def aiml_admin():
    print("AIML Department Admin: Dr. Kavitha")

def aiml_cabin():
    print("AIML HOD Cabin: Block C – Room 305")


# -------- CSBS Module --------
def csbs_admin():
    print("CSBS Department Admin: Dr. Arun")

def csbs_cabin():
    print("CSBS HOD Cabin: Block D – Room 210")
```

```python
# -------- Main Program --------
print("=== AI & DS Department ===")
ai_ds_admin()
ai_ds_cabin()

print("\n=== CSE Department ===")
cse_admin()
cse_cabin()

print("\n=== AIML Department ===")
aiml_admin()
aiml_cabin()

print("\n=== CSBS Department ===")
csbs_admin()
csbs_cabin()
```

Output

Inference:

Viva-Voce:

1) **What is meant by a package in Python?**

2) **How does this single program logically represent packages and modules?**

3) **What is the purpose of defining separate functions for each department?**

4) **How would this program differ if implemented using actual packages and sub-packages?**

5) **What is the advantage of modular programming in Python?**

Result:

Ex.No11a)	**BOUNCING BALL USING PYGAME**
Date :	

Aim

To write a Python program using Pygame that simulates the motion of a bouncing ball inside a window, reversing direction whenever it hits the boundary.

Algorithm

Step 1: Import the Pygame and initialize it.

Step 2: Create a display window.

Step 3: Set the initial ball position, radius, speed, and direction.

Step 4: Run a game loop until the user quits.

Step 5: Move the ball by updating X and Y coordinates.

Step 6: Check collision with window borders.

Step 7: Reverse ball direction upon collision to simulate bounce.

Step 8: Clear screen and redraw ball continuously.

Step 9: Update the display and control frame rate.

Program

```
import pygame

pygame.init()

# Window setup
width, height = 600, 400
screen = pygame.display.set_mode((width, height))
pygame.display.set_caption("Bouncing Ball Simulation")

# Ball properties
x, y = 100, 100
radius = 20
speed_x = 4
speed_y = 3

clock = pygame.time.Clock()
running = True

while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Ball movement
    x += speed_x
    y += speed_y

    # Collision detection
    if x - radius <= 0 or x + radius >= width:
        speed_x *= -1
    if y - radius <= 0 or y + radius >= height:
        speed_y *= -1
```

```
        screen.fill((0, 0, 0))
        pygame.draw.circle(screen, (255, 0, 0), (x, y), radius)

        pygame.display.update()
        clock.tick(60)

    pygame.quit()
```

Output

Inference:

Result:

Ex.No:11b)	**ELLIPTICAL ORBIT USING PYGAME**
Date:	

Aim

To create a Pygame program that simulates a planet revolving around the sun in an elliptical orbit.

Algorithm

Step 1: Import and initialize Pygame.

Step 2: Create a display window.

Step 3: Place a stationary object (sun) at the center.

Step 4: Initialize angle and orbit radii for elliptical path.

Step 5: In game loop, increase angle and calculate planet position using ellipse formula:

x=centerx+a·cos(θ)x = center_x + a \cdot \cos(\theta)x=centerx+a·cos(θ)

y=centery+b·sin(θ)y = center_y + b \cdot \sin(\theta)y=centery+b·sin(θ)

Step 6: Redraw sun and moving planet continuously.

Step 7: Update display until window is closed.

Program

```
import pygame
import math

pygame.init()

width, height = 700, 500
screen = pygame.display.set_mode((width, height))
pygame.display.set_caption("Elliptical Orbit Simulation")

center_x, center_y = width // 2, height // 2
a = 200  # horizontal radius (major axis)
b = 120  # vertical radius (minor axis)
angle = 0

clock = pygame.time.Clock()
running = True

while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    angle += 0.02  # increase angle to simulate orbit

    x = center_x + int(a * math.cos(angle))
    y = center_y + int(b * math.sin(angle))

    screen.fill((0, 0, 20))

    # Draw Sun
    pygame.draw.circle(screen, (255, 255, 0), (center_x, center_y), 30)

    # Draw Planet
    pygame.draw.circle(screen, (0, 150, 255), (x, y), 15)

    pygame.display.update()
    clock.tick(60)
```

```
pygame.quit()
```

Output

Inference:

Viva-Voce:

1) **What is Pygame and for what type of applications is it mainly used?**

2) **Which Pygame module is used to initialize the game environment and why is it required?**

3) What is the purpose of pygame.display.update() or pygame.display.flip()?

4) What is the role of the game loop in a Pygame program?

5) How are keyboard or mouse events handled in Pygame?

Result:

Ex.No:12	**FILE HANDLING**
Date :	

Aim

To write a Python program to perform read and write operations on a text file and find the number of occurrences of a given word in the file.

Algorithm

Step 1: Start the program.

Step 2: Open a text file in write mode and write some data into it.

Step 3: Close the file.

Step 4: Reopen the same file in read mode.

Step 5: Read the contents of the file.

Step 6: Ask the user to input a word to search.

Step 7: Split the file text into individual words.

Step 8: Count how many times the given word appears in the text.

Step 9: Display the total number of occurrences.

Step 10: End the program.

Program

```
# Program to perform read and write operations in a file
# and find the occurrence of a given word

# Writing to a file
file = open("sample.txt", "w")
file.write("Python is easy to learn.\n")
file.write("Python is powerful.\n")
file.write("Learning Python is fun.\n")
file.close()

# Reading from the file
file = open("sample.txt", "r")
content = file.read()
file.close()

print("File Content:\n")
print(content)
```

```
# Finding occurrence of a given word
word = input("Enter the word to find: ")

count = 0
words = content.split()

for w in words:
    if w.lower() == word.lower():
        count += 1

print(f"\nThe word '{word}' occurs {count} times in the file.")
```

Output

Inference:

Viva-Voce:

1) **What is the difference between file write mode ("w") and append mode ("a") in Python?**

2) **How does Python read data from a text file?**

3) **Why is it important to close a file after reading or writing?**

4) **How is the occurrence of a word identified in the file content?**

5) **How does this program handle case sensitivity while searching for a word?**

Result:

Ex.No:13	**ARRAY OPERATIONS**
Date :	

Aim

To write a Python program to perform various array operations such as insertion, deletion, searching, updating, and displaying elements using Jupyter / Lambda Notebook.

Algorithm

Step 1: Start the program.

Step 2: Create an empty array.

Step 3: Display a menu with operations:

Step 4: Insert an element

Step 5: Delete an element

Step 6: Search an element

Step 7: Update an element

Step 8: Display array

Step 9: Read user choice.

Step 10: Perform the selected operation:

- Insert → append the element to the array
- Delete → remove the element if present
- Search → check if the element exists
- Update → modify the value at a specified index
- Display → print array elements

Step 11: Repeat until the user chooses exit.

Step 12: End the program.

Program

```
# Program for Array Operations

arr = []  # empty array

while True:
    print("\n--- ARRAY OPERATIONS MENU ---")
    print("1. Insert Element")
    print("2. Delete Element")
    print("3. Search Element")
    print("4. Update Element")
    print("5. Display Array")
    print("6. Exit")

    choice = int(input("Enter your choice: "))

    if choice == 1:
        ele = int(input("Enter element to insert: "))
        arr.append(ele)
        print("Element inserted successfully.")

    elif choice == 2:
        ele = int(input("Enter element to delete: "))
        if ele in arr:
            arr.remove(ele)
            print("Element deleted.")
        else:
            print("Element not found.")

    elif choice == 3:
        ele = int(input("Enter element to search: "))
        if ele in arr:
            print("Element found at index:", arr.index(ele))
        else:
            print("Element not found.")

    elif choice == 4:
        index = int(input("Enter index to update: "))
        if 0 <= index < len(arr):
            new_val = int(input("Enter new value: "))
            arr[index] = new_val
            print("Value updated.")
        else:
            print("Invalid index.")
```

```
    elif choice == 5:
        print("Array elements:", arr)

    elif choice == 6:
        print("Exiting program.")
        break

    else:
        print("Invalid choice. Try again.")
```

Output

Inference:

Viva-Voce:

1) **What is an array in Python and how is it different from a list?**

2) **Why is the array module used instead of lists in some cases?**

3) How do you insert and delete elements in an array?

4) How is searching performed in a Python array?

5) What does the len() function return when applied to an array?

Result:

Ex.No:14	**SORT NUMBERS AND STRINGS**
Date:	

Aim

To write a Python program to perform array operations such as sorting a list of numbers and sorting a list of strings using Jupyter / Lambda Notebook.

Algorithm

Step 1: Start the program.

Step 2: Create a list of numbers and a list of strings.

Step 3: Print the original lists.

Step 4: Sort the list of numbers in ascending and descending order.

Step 5: Sort the list of strings in alphabetical and reverse alphabetical order.

Step 6: Display the sorted results.

Step 7: End the program.

Program

```
# Program to sort numbers and strings

# List of numbers
numbers = [42, 10, 85, 23, 16, 5]
print("Original Numbers:", numbers)

# Sorting numbers
asc_numbers = sorted(numbers)
desc_numbers = sorted(numbers, reverse=True)

print("Numbers Sorted Ascending:", asc_numbers)
print("Numbers Sorted Descending:", desc_numbers)

# List of strings
```

```
names = ["Rahul", "Anitha", "Priya", "Kishore", "Bala"]
print("\nOriginal Strings:", names)

# Sorting strings
asc_strings = sorted(names)
desc_strings = sorted(names, reverse=True)

print("Strings Sorted Alphabetically:", asc_strings)
print("Strings Sorted Reverse Alphabetically:", desc_strings)
```

Output

Inference:

Viva-Voce:

1) **What is the difference between sorting numbers and sorting strings in Python?**

2) Which method is used to sort a list in Python and does it modify the original list?

3) Why are lists used for sorting instead of arrays from the array module?

4) How does Python determine the order while sorting strings?

5) What are the common operations that can be performed on arrays in Python?

Result: